

TinyDet: Accurate Small Object Detection in Lightweight Generic Detectors*

Shaoyu Chen¹, Tianheng Cheng¹, Jiemin Fang², Qian Zhang³, Yuan Li⁴, Wenyu Liu¹, Xinggang Wang^{1,†}

¹ School of EIC, Huazhong University of Science and Technology

² Institute of AI, Huazhong University of Science and Technology

³ Horizon Robotics ⁴ Google

Source code and pretrained models are available at: hustvl/TinyDet

Abstract

Small object detection requires the detection head to scan a large number of positions on image feature maps, which is extremely hard for computation- and energy-efficient lightweight generic detectors. To accurately detect small objects with limited computation, we propose a two-stage lightweight detection framework with extremely low computation complexity, termed as TinyDet. It enables high-resolution feature maps for dense anchoring to better cover small objects, proposes a sparsely-connected convolution for computation reduction, enhances the early stage features in the backbone, and addresses the feature misalignment problem for accurate small object detection. On the COCO benchmark, our TinyDet-M achieves 30.3 AP and 13.5 AP^s with only 991 MFLOPs, which is the first detector that has an AP over 30 with less than 1 GFLOPs; besides, TinyDet-S and TinyDet-L achieve promising performance under different computation limitation.

1. Introduction

Object detection plays an important role in computer vision, and gradually becomes the technical foundation of many applications, such as autonomous driving, remote sensing and video surveillance. However, the inference procedure of advanced detection models costs massive computational resources, which makes them hard to be applied on resource-constrained mobile or edge devices. To widely apply artificial intelligence (AI), “Tiny AI” models that are both computation-efficient and energy-efficient are becoming more and more popular. In this paper, we design a lightweight generic object detection framework, termed as

* A letter version of this paper is published in SCIENCE CHINA Information Sciences (SCIS) with doi.org/10.1007/s11432-021-3504-4. Please cite the SCIS version. † Corresponding author: Xinggang Wang (xgwang@hust.edu.cn).

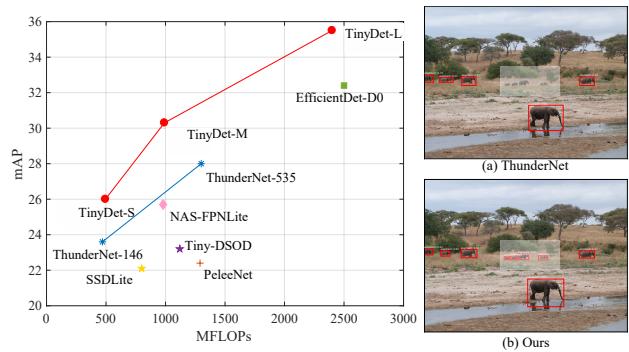


Figure 1. Model FLOPs vs COCO Accuracy. Our TinyDet achieves higher mAP with less computation cost compared with other detectors.

TinyDet, for efficient and accurate object detection, especially small objects, with low computation cost.

In recent years, many innovative and representative lightweight detection models [6, 34, 38, 41, 44] have been proposed for the better trade-off between the computation cost and accuracy. To reduce the computation cost, they usually downscale the input image by a large ratio and perform object detection on small feature maps. For example, Pelee [44] takes 304×304 input and the largest feature map used for detection in Pelee is 19×19 . ThunderNet [34] takes 320×320 input, and only uses a single feature map with the resolution of 20×20 for detection. For comparison, large models like Faster R-CNN [37] with feature pyramid network (FPN) [23] takes 800×1333 input and the largest feature map is as large as 200×333 . Performing object detection with small input images and on small feature maps is helpful to reducing computation cost. However, small feature maps have no detailed information and poor positional resolution. Previous lightweight detectors have very limited ability to detect small objects. They sacrifice the detection performance of small objects for high efficiency.

The capacity of detecting small objects is of great importance for object detection based applications and is a key factor for evaluating an object detection model. As demonstrated in the COCO dataset, approximately 41% objects are small (area < 32^2) [25]. In this paper, we target at boosting the small object detection performance in lightweight generic detection networks. Based on the good practices of designing lightweight networks, we propose TinyDet, which is a two-stage detector with high-resolution (HR) feature maps for dense anchoring. HR feature maps significantly improve the small object detection ability but also bring much more computation cost. To release the contradiction between resolution and computation, we propose TinyFPN and TinyRPN by introducing a sparsely-connected convolution (SCConv), which keeps both high resolution and low computation. We improve the backbone network for better small object detection performance. Small object detection relies more on detailed information in shallow features. We keep more detailed information by allocating more computation to the early stages. Besides, we observe that severe feature misalignment exists in lightweight detectors. The feature misalignment accumulates layer by layer and is passed to the detection part, affecting the precision of regression in both RPN and RCNN head. Small objects are much more sensitive to such positional misalignment. By eliminating the misalignment, the detection performance of small objects is significantly improved.

Our contributions can be summarized as:

- In lightweight detection networks, for the first time, we enable the high-resolution detection feature map (*i.e.*, 80×80) for dense anchoring, which is essential for detecting small objects.
- We propose TinyFPN and TinyRPN with the sparsely-connected convolution to perform efficient object detection with high-resolution detection feature maps.
- By enhancing the early stages in the backbone and addressing the misalignment problem in TinyDet, we further improve the detection results of small objects.
- Our TinyDet models have strong performance and little computation budget on the COCO *test-dev2017* set as shown in Fig. 1. TinyDet-M achieves 30.3 mAP with only 991 MFLOPs, which is the state-of-the-art result among lightweight detectors. Notably, the small object detection performance is outstanding, the APs of TinyDet-S and TinyDet-M are two times of that of ThunderNets.

2. Related Work

Lightweight Object Detector Lightweight models for generic object detection has witnessed rapid developments

in recent years. Firstly, advances of lightweight classification network design methods [14, 15, 30, 38, 49] directly boost the development of lightweight object detection. The lightweight classification networks are often directly adopted as backbones of detectors to extract features, *e.g.*, the hand-crafted MobileNetV2 [38] is used in SSDLite [38] and the neural architecture searched EfficientNet [40] is used in EfficientDet [41]. But detection and classification need different backbones [22]. To better match characteristics of detection tasks, in many lightweight detectors [6, 34], specialized backbones are proposed, based on existing classification networks. Secondly, well developed pipelines [23, 24, 27, 37] for object detection also build a solid foundation for lightweight detector research. Most lightweight detectors [36, 38, 41, 44] follow the compact one-stage architecture. PeleeNet [44] is only built with conventional convolutions without using the popular mobile convolution. RefineDetLite [6] makes design specialized for CPU-only devices. EfficientDet [41] proposes a weighted bi-directional feature pyramid network for easy and fast feature fusion and builds a scalable detection architecture across a wide spectrum of resource constraints. Two-stage detectors, with more complicated pipelines, are usually thought to be more time consuming in inference phase. However, some [9, 21, 34] prove that two-stage detectors can also be as efficient as one-stage ones if the second stage is made lightweight enough. The two-stage paradigm tends to be better performing at detecting small objects. Thus, we follow the two-stage paradigm to design our detector by considering both efficiency and accuracy.

Small Object Detection Detecting small objects from the video and image has a high-profile in computer vision, remote sensing, autonomous driving, *etc.*. Liu *et al.* [27] creates more small object training examples by reducing the size of large objects. D-SSD [10], C-SSD [48] F-SSD [5] and ION [2] focus on building appropriate context features for small object detection. Hu *et al.* [17] makes use of a coarse image pyramid and uses two times upsampled input images to detect small faces. Several studies, such as [1, 19, 31], use generative adversarial network (GAN) [13] to generate super-resolved features for small object detection. Larger input resolution and super-resolution methods bring much more computation cost and are not suitable for lightweight detector design. Our work is slightly related to high-resolution network (HRNet) [45], which maintains a high-resolution representation in the whole network for position-sensitive visual recognition tasks. We find that there are no papers that focus on detecting small objects in lightweight generic detectors. In this paper, we target at accurate small object detection while keeping low computation budget.

Sparse Convolution Reducing the redundancy in deep convolutional neural networks is an important direction to explore when designing lightweight detection networks. Many previous works dedicate to reduce the redundancy by making the connections in convolution sparser. Liu *et al.* [26] adopts sparse decompositions to get sparse convolutional networks and lower the computational complexity. The depth-wise separable convolution [8, 15], a kind of factorized convolution, is widely adopted for reducing computation and model size. And the group convolution is used in [49] for efficient model design. Our work combines the depth-wise convolution and group convolution in a novel form to build extremely lightweight FPN and RPN.

3. Method

In this section, we detail the design of TinyDet and illustrate how a generic detector with low FLOPs performs accurate small object detection. We follow the two-stage detection paradigm, which is more friendly to small object detection [27]. The detector contains four parts: the enhanced backbone, TinyFPN, TinyRPN and R-CNN head. For clarity, we only provide details of TinyDet-M in this section and leave the details of TinyDet-S and TinyDet-L in Appendix.

3.1. Backbone Network

3.1.1 Detailed Information Enhancement

Early-stage feature maps with high resolution in the backbone contain abundant detailed information, which is vital for recognizing and localizing small objects. Existing lightweight backbone networks [14, 15, 30, 38, 49] usually downsample feature maps rapidly, keeping less layers and channels in high resolution stages. This setting successfully minimizes computational budget but sacrifices much detailed information.

To improve the performance of detecting small objects, we propose a detailed information enhanced backbone based on the high-performance network, MobileNetV3 [14]. Tab. 1 shows the detailed network configuration. Compared with other widely used lightweight backbone networks [14, 15, 30, 38, 49], we allocate more computation to the early stages with higher resolution. With this design, more detailed information is extracted and kept for detecting small objects.

3.1.2 Solving the Feature Alignment Problem

Convolution with stride 2 is widely used to reduce the resolution of the feature maps [14, 15, 30, 38, 49]. And in detection, even input resolution is a common practice. Because odd input pixels ruin the proportional relation between different pyramid levels, making the coordinate mapping be-

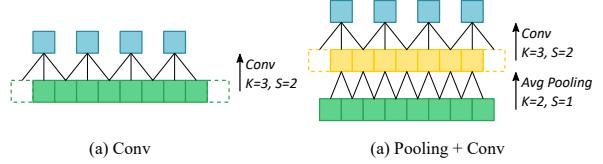


Figure 2. The feature alignment problem. (a) When the number of input pixels is even, due to spatial asymmetry, the strided convolution causes a 0.5-pixel misalignment. (b) By introducing an average pooling before the strided convolution, even pixels are converted to odd ones, eliminating asymmetry and misalignment. Dashed squares denote zero padding.

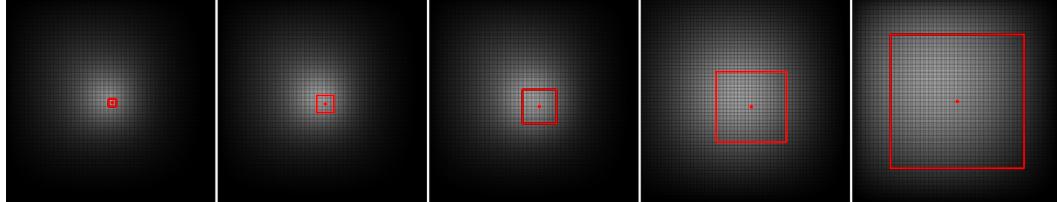
tween input and output complex. However, strided convolutions on even input resolution may lead to feature misalignment. As shown in Fig. 2(a), considering a convolution layer with stride 2 and the input resolution is even, during the convolution computation, one padding pixel is ignored, resulting in spatial asymmetry and 0.5 pixel feature misalignment. The feature misalignment caused by several strided convolutions is accumulated layer by layer through the whole network and becomes more significant in higher levels. The misalignment is negligible for large models with high input resolution, but significant for lightweight detectors with low resolution. It severely degrades the performance. The ROI pooling/align operation would extract misaligned features for every object proposals. Small object detection requires more accurate localization and are affected more.

To alleviate the feature misalignment in TinyDet, we adopt average pooling layers before each strided convolution. As illustrated in Fig. 2(b), the average pooling operation converts even pixels to odd ones and avoids asymmetry in strided convolution and correct the misalignment. And when optimizing the network, the adjacent convolution layer and averaging pooling layer can be fused into a single layer for better inference efficiency.

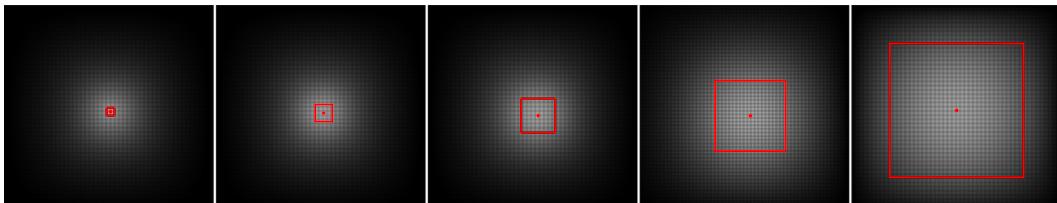
Theoretical Calculation of Feature Misalignment We provide the theoretical calculation about the feature misalignment caused by strided convolution. For a convolution with stride 2, we assume that the kernel size is k , the padding size is $\frac{k-1}{2}$, and the input feature map is with stride s compared with the input image. As shown in Fig. 2(a), one padding pixel is dropped, and the feature center is shifted by 0.5 pixel. When mapped to the input image, the misalignment is $\frac{s}{2}$ pixels. The feature misalignment affects subsequent layers and is accumulated layer by layer. In TinyDet, 6 strided convolution layers exist in the backbone network and respectively cause misalignment of 0.5, 1, 2, 4, 8, 16 pixels. In FPN and RPN, the accumulated misalignment is as large as 31.5 pixels. For lightweight de-

Table 1. Specification for the backbone of TinyDet-M. Bneck denotes the inverted residual bottleneck structure [38]. ExSize denotes expansion size. SE denotes the squeeze-and-excitation module [16]. NL denotes the type of the used nonlinearity. HS denotes h-swish [14] and RE denotes ReLU. FPN denotes whether the output of the block is fed into FPN. Enhancement denotes modifications compared with the original MobileNetV3 and "c" denotes channel.

| Input | Operator | ExSize | Out | SE | NL | Stride | FPN | Enhancement |
|-------------------|----------------------|--------|-----|----|----|--------|-----|--------------|
| $320^2 \times 3$ | conv2d, 3×3 | - | 24 | - | HS | 2 | - | c + 50% |
| $160^2 \times 24$ | bneck, 3×3 | 24 | 24 | - | RE | 1 | - | c + 50% |
| $160^2 \times 24$ | bneck, 3×3 | 72 | 36 | - | RE | 2 | - | c + 50% |
| $80^2 \times 36$ | bneck, 3×3 | 108 | 36 | - | RE | 1 | - | added blocks |
| $80^2 \times 36$ | bneck, 3×3 | 108 | 36 | - | RE | 1 | - | added blocks |
| $80^2 \times 36$ | bneck, 3×3 | 108 | 36 | - | RE | 1 | ✓ | c + 50% |
| $80^2 \times 36$ | bneck, 5×5 | 108 | 60 | ✓ | RE | 2 | - | c + 50% |
| $40^2 \times 60$ | bneck, 5×5 | 180 | 60 | ✓ | RE | 1 | - | c + 50% |
| $40^2 \times 60$ | bneck, 5×5 | 180 | 60 | ✓ | RE | 1 | ✓ | c + 50% |
| $40^2 \times 60$ | bneck, 3×3 | 240 | 80 | - | HS | 2 | - | - |
| $20^2 \times 80$ | bneck, 3×3 | 200 | 80 | - | HS | 1 | - | - |
| $20^2 \times 80$ | bneck, 3×3 | 184 | 80 | - | HS | 1 | - | - |
| $20^2 \times 80$ | bneck, 3×3 | 184 | 80 | - | HS | 1 | - | - |
| $20^2 \times 80$ | bneck, 3×3 | 480 | 112 | ✓ | HS | 1 | - | - |
| $20^2 \times 112$ | bneck, 3×3 | 672 | 112 | ✓ | HS | 1 | ✓ | - |
| $20^2 \times 112$ | bneck, 5×5 | 672 | 160 | ✓ | HS | 2 | - | - |
| $10^2 \times 160$ | bneck, 5×5 | 960 | 160 | ✓ | HS | 1 | - | - |
| $10^2 \times 160$ | bneck, 5×5 | 960 | 160 | ✓ | HS | 1 | ✓ | - |
| $10^2 \times 160$ | bneck, 5×5 | 960 | 160 | ✓ | HS | 2 | - | - |
| $5^2 \times 160$ | bneck, 5×5 | 960 | 160 | ✓ | HS | 1 | ✓ | - |



(a) Without feature alignment



(b) With feature alignment

Figure 3. Visualization of ERFs of different pyramid levels. From left to right: pyramid levels with stride 4, 8, 16, 32 and 64 in FPN. The red point denotes the geometry center. The red square indicates the corresponding anchor size in each pyramid level.

tectors with small input size, the misalignment is quite significant. Considering the input resolution of 320×320 , the misalignment proportion is up to $\frac{31.5}{320} \approx 9.8\%$, which leads to severe mismatching between features and their spatial positions.

Visualization of Effective Receptive Field To better demonstrate the effect of feature misalignment, we adopt effective receptive field (ERF) maps [29] to visualize the misalignment of feature maps. As shown in Fig. 3(a), we can observe that ERFs obviously deviate from the geometric centers of corresponding anchors. Pixel-level prediction in RPN and region-based feature extraction in R-CNN would

be based on misaligned features. With average pooling layers applied, the misalignment is eliminated (Fig. 3(b)).

3.2. TinyFPN and TinyRPN

3.2.1 High Resolution Detection Feature Maps

Under the limitation of computation cost, previous lightweight detectors usually adopt feature maps with low resolutions for detection (38×38 in SSDLite [38], 19×19 in Pelee [44], 20×20 in ThunderNet [34]). However, small feature maps have low spatial resolution. Low-resolution feature maps cannot provide spatially matched features for objects located in arbitrary positions, especially for small objects. In this paper, we enable object detection on high-resolution feature maps. We fetch five feature maps from the backbone for detection, respectively with stride 4, 8, 16, 32 and 64. Note that the resolution of the feature map with stride 4 is 80×80 , which is the highest resolution feature map used in lightweight detectors. More analysis on the importance of the high-resolution setting can be found in Sec. 3.3.

3.2.2 Sparsely-connected Convolution for Computation Reduction

Due to the high-resolution design, computation budget of the detection part becomes extremely high. Though the depth-wise separable convolution [8, 15] has been widely used for detection part design in prior lightweight detectors [6, 34, 41] to reduce the computation cost, we find it not enough in our high-resolution setting. Consequently, we exploit a sparsely-connected convolution (SCConv), specialized for both efficiency and high resolution in FPN and RPN. As shown in Fig. 4, the SCConv is a combination of a depth-wise convolution [15] and a point-wise group convolution [49]. Compared with the vanilla depth-wise separable convolution [8, 15], SCConv further reduces the connections among channels. Experiments in Sec. 4.3 shows that this sparser setting has little influence on detection performance and reduces the computation cost by a large amount.

Based on SCConv, we propose TinyFPN and TinyRPN. In TinyFPN (Fig. 4), SCConv is applied after feature fusion, in place of the normal 3×3 convolutions. We set larger group numbers, *i.e.* sparser connections, for SCConvs in the high-resolution pyramid levels to reduce the computation cost. TinyRPN (Fig. 4) consists of a SCConv and two sibling 1×1 convolutions for classification and regression respectively. The parameters of TinyRPN are shared across all pyramid levels. Ablation studies about the group settings of SCConvs are provided in Sec. 4.3.

3.3. High-resolution Feature Maps for Dense Anchoring

Translation variance is a challenge in object detection. Detectors should be insensitive to translation and deal with objects located in arbitrary positions. The introduction of anchor [37] eases the problem of translation variance in object detection. A large number of anchors are evenly tiled over the whole image and each anchor is only responsible for predicting objects that appear in a certain region, which is defined as *responsive region*. During training, objects are assigned to anchors according to their IoUs and an assignment strategy [37]. However, two problems emerge in the procedure of anchor assignment when anchors are not spatially dense enough, which are described as follows.

- With the assignment IoU threshold fixed, responsive regions are fixed. As is shown in Fig. 5(a), when anchors are not dense enough, responsive regions cannot cover the whole image. We call the uncovered region *overlooked region*. Objects, especially small objects, located in overlooked regions are never assigned to any anchor during training. Consequently, in the inference phase, these objects are less possible to be detected for lack of anchors here.
- If we lower the assignment IoU threshold, the responsive regions are enlarged while the overlooked regions shrink or even vanish. But enlarged responsive regions make it hard for detectors to obtain accurate results. Each anchor deals with more objects and more variance in object shape and position, which is illustrated in Fig. 5(b).

As discussed above, the anchor density should be high enough to cover possible objects. Thus, in TinyDet, we keep high-resolution feature maps for dense anchoring. To comprehend how dense anchoring impact detection performance, especially for small objects, we take TinyDet with ThunderNet [34] for comparison. In Fig. 6, we visualize the distribution of smallest anchors. In ThunderNet, the distance between adjacent anchors is 16 pixels. It's hard to match small objects with anchors spatially. While that in TinyDet is only 4 pixels. Densely tiled anchors better cover small objects.

Table 2. Ground-truth miss-assignment ratio (GTMR) for Faster R-CNN, ThunderNet and our TinyDet. GTMR^s, GTMR^m and GTMR^l for small, medium and large objects respectively.

| Detector | GTMR(%) | GTMR ^s (%) | GTMR ^m (%) | GTMR ^l (%) | Input |
|---------------------|---------|-----------------------|-----------------------|-----------------------|-------------------|
| Faster R-CNN | 6.6 | 17.0 | 1.2 | 1.9 | 800×1333 |
| Faster R-CNN w/ FPN | 6.1 | 15.2 | 1.2 | 2.3 | 800×1333 |
| ThunderNet | 18.1 | 53.1 | 5.6 | 1.2 | 320×320 |
| TinyDet | 8.4 | 21.2 | 2.2 | 3.0 | 320×320 |

Quantitatively, we propose *ground-truth miss-assignment ratio* (GTMR) to evaluate the assignment

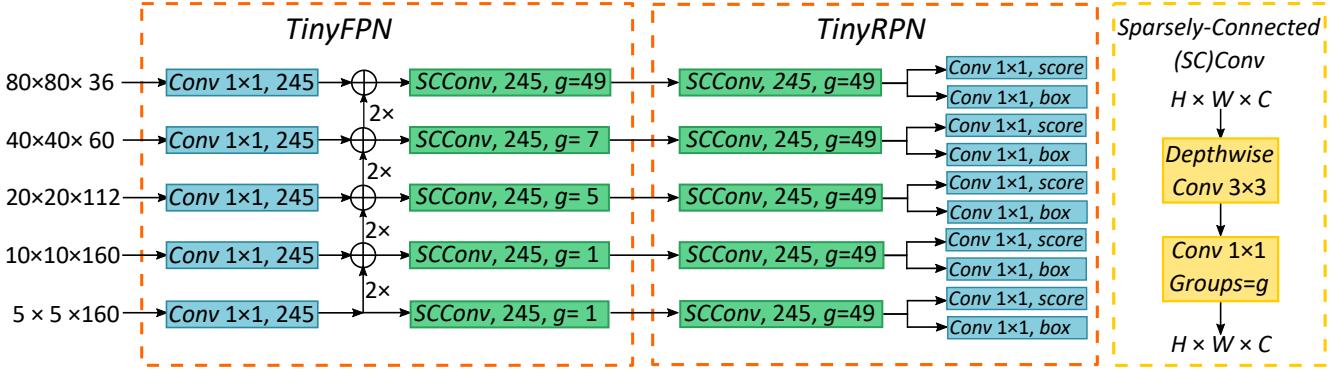


Figure 4. TinyFPN and TinyRPN with sparsely-connected convolutions (SCConvs). SCConv consists of a depth-wise convolution and a point-wise group convolution. "g" denotes the number of groups in SCConv. 245 is the number of output channels for both vanilla $Conv\ 1 \times 1$ and SCConv.

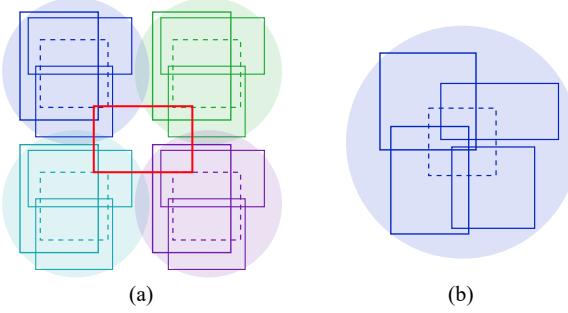


Figure 5. Problems caused by sparse anchors. Dashed squares denote anchors. Solid rectangles denote possible objects that may exist in an image. (a): Objects located in the overlooked regions (the red box) have low IoUs with every anchor, and would not be assigned to any anchor during training. (b): With the responsive region enlarged, every anchor deals with more objects and more variance in object shape and position. Detection becomes harder.

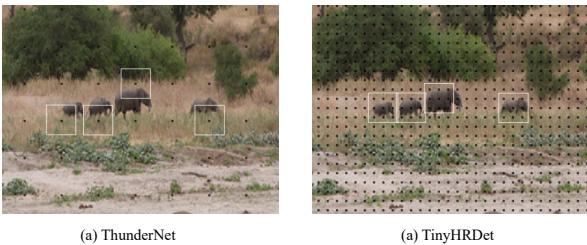


Figure 6. Anchor density comparison between ThunderNet and TinyDet. Black points denote centers of evenly tiled anchors and white boxes show the anchors that have the best IoU with a ground-truth object. Densely tiled anchors better cover small objects.

procedure. GTMR is defined as the proportion of those ground-truth objects that are not assigned to any anchor

over all the objects. It reflects the matching quality between anchors and objects under a certain assignment strategy. As shown in Tab. 2, GTMR of ThunderNet is up to 18.1%, much higher than that of Faster R-CNN [37]. TinyDet obtains a quite lower GTMR, 8.4%, though as lightweight as ThunderNet. Notably, GTMR of small objects of ThunderNet is extremely high; while, in our TinyDet with high-resolution feature maps and densely tiled anchors, GTMR of small objects is much lower than ThunderNet. Small objects can be better matched with anchors spatially.

4. Experiments

In this section, we first describe the experimental details. Then we compare our TinyDet models with other state-of-the-art (SOTA) methods. We further perform detailed ablation studies to demonstrate the effectiveness of our proposed methods.

4.1. Experimental Details

Our experiments are conducted on the COCO dataset [25]. We use the *train2017* split for training and report our main results on the *test-dev2017*. The *val2017* split is used for detailed ablation studies. We follow the standard COCO detection metrics to report the average precision (AP) under different IoUs (*i.e.*, AP^{50} , AP^{75} and the overall AP) and AP for detecting objects in different scales (*i.e.*, AP^s , AP^m and AP^l).

We implement our TinyDet based on the PyTorch [32] framework and MMDetection [7] toolbox. Our models are trained with batch size 128 on 4 GPUs (32 images per GPU) for 240 epochs. The SGD optimizer is used with momentum 0.9 and weight decay 1e-5. We linearly increase the learning rate from 0 to 0.35 in the first 500 iterations and then decay it to 1e-5 using the cosine anneal schedule [28]. 2000/200 proposals are used in the second stage at the training/inference phase. We adopt the same data augmenta-

Table 3. SOTA results on the COCO *test-dev2017* set. All the results are obtained via the single-model and single-scale testing. [†] means the model is re-implemented by ourselves. [‡] denotes the result is evaluated on *val2017* set because the result on *test-dev2017* set is not provided.

| Method | FLOPs | AP | AP ⁵⁰ | AP ⁷⁵ | AP ^s | AP ^m | AP ^l | Input |
|---------------------------------|-------|------|------------------|------------------|-----------------|-----------------|-----------------|------------------|
| ThunderNet-SNet146 [34] | 470M | 23.6 | 40.2 | 24.5 | - | - | - | 320 ² |
| ThunderNet-SNet146 [†] | 499M | 23.8 | 40.5 | 24.7 | 4.6 | 23.0 | 42.9 | 320 ² |
| TinyDet-S | 495M | 26.0 | 45.8 | 26.5 | 9.6 | 26.8 | 39.5 | 320 ² |
| MobileNetV2-SSDLite [38] | 800M | 22.1 | - | - | - | - | - | 320 ² |
| MobileNet-SSDLite [38] | 1300M | 22.2 | - | - | - | - | - | 320 ² |
| Pelee [44] | 1290M | 22.4 | 38.3 | 22.9 | - | - | - | 304 ² |
| Tiny-DSOD [20] | 1120M | 23.2 | 40.4 | 22.8 | - | - | - | 300 ² |
| YOLOX-Nano [‡] [11] | 1080M | 25.3 | - | - | - | - | - | 416 ² |
| NASFPNLite MobileNetV2 [12] | 980M | 25.7 | - | - | - | - | - | 320 ² |
| ThunderNet-SNet535 [34] | 1300M | 28.0 | 46.2 | 29.5 | - | - | - | 320 ² |
| ThunderNet-SNet535 [†] | 1297M | 27.3 | 45.4 | 28.4 | 6.5 | 28.4 | 46.2 | 320 ² |
| TinyDet-M | 991M | 30.3 | 51.2 | 31.8 | 13.5 | 30.9 | 43.9 | 320 ² |
| YOLOv2 [35] | 17.5G | 21.6 | 44.0 | 19.2 | 5.0 | 22.4 | 35.5 | 416 ² |
| PRN [42] | 4.0G | 23.3 | 45.0 | 22.0 | 6.7 | 24.8 | 35.1 | 416 ² |
| EFM (SAM) [43] | 5.1G | 26.8 | 49.0 | 26.7 | 9.8 | 28.2 | 38.8 | 416 ² |
| YOLOX-Tiny [‡] [11] | 6.5G | 31.7 | - | - | - | - | - | 416 ² |
| EfficientDet-D0 [41] | 2.5G | 32.4 | - | - | - | - | - | 512 ² |
| YOLOv3 [36] | 71.0G | 33.0 | 57.9 | 34.4 | 18.3 | 35.4 | 41.9 | 608 ² |
| YOLOv5 [18] | 17.0G | 36.7 | - | - | - | - | - | 640 ² |
| TinyDet-L | 2.4G | 35.5 | 56.8 | 38.3 | 18.3 | 37.5 | 48.4 | 512 ² |

tion strategy introduced in SSD [27]. Following ThunderNet [34], online hard example mining (OHEM) [39], Soft-NMS [3], and Cross-GPU Batch Normalization [33] are used.

In the proposed TinyRPN head, anchors in five pyramid levels are respectively with sizes of 12.8^2 , 25.6^2 , 51.2^2 , 102.4^2 and 204.8^2 . Anchors of multiple aspect ratios $\{1 : 2, 1 : 1, 2 : 1\}$ are used in each level, the same as [37] and [23]. As for the R-CNN head, we adopt position-sensitive RoI align [9, 21, 34] to extract box features effectively.

4.2. Comparison with SOTA Lightweight Generic Detectors

We compare our TinyDet with other SOTA lightweight detectors on the COCO *test-dev2017* set in Tab. 3. Among the compared methods, ThunderNet [34] and EfficientDet [41] are regarded as the most recent SOTA lightweight detectors. The results show that our models obviously outperform them with fewer computation costs: TinyDet-S surpasses ThunderNet-SNet146 by 2.4% AP with similar FLOPs; TinyDet-M surpasses ThunderNet-SNet535 by 2.3% AP with 76% FLOPs; and TinyDet-L surpasses EfficientDet-D0 by 3.1% AP with similar FLOPs. Besides, TinyDet has better performance-computation trade-offs than the automatically searched lightweight detec-

tor (*i.e.*, NAS-FPNLite [12]) and the popular YOLO series [18, 35, 36].

Our TinyDet models obtain extraordinary performance on detecting small objects. With similar computation cost (*i.e.*, FLOPs), TinyDet-M achieves 13.5 AP^s, which is over 100% improvement over ThunderNet-SNet535 with 6.5 AP^s; and TinyDet-S achieves 9.6 AP^s, which is also over 100% improvement over ThunderNet-SNet146 with 4.6 AP^s. TinyDet-L achieves the same 18.3 AP^s with YOLOv3 [36], but it only has 1/30 computation cost of YOLOv3. Some visualized results are shown in Fig. 7.

4.3. Ablation Studies

4.3.1 Ablation Study on Backbone Enhancement

As discussed in Sec. 3.1.1, we enhance detailed information in the MobileNetV3-based backbone, which is important for improving the feature representation of high resolution. We provide three MobileNetV3’s variants: MobileNetV3-B, -C and -BC to demonstrate our effectiveness of the enhanced backbone. Compared with original MobileNetV3, MobileNetV3-B adds 2 extra blocks in the stage with stride 4 and MobileNetV3-C contains 50% more channels in early stages. MobileNet-BC is the combination of MobileNetV3-B and MobileNetV3-C which is our proposed backbone for

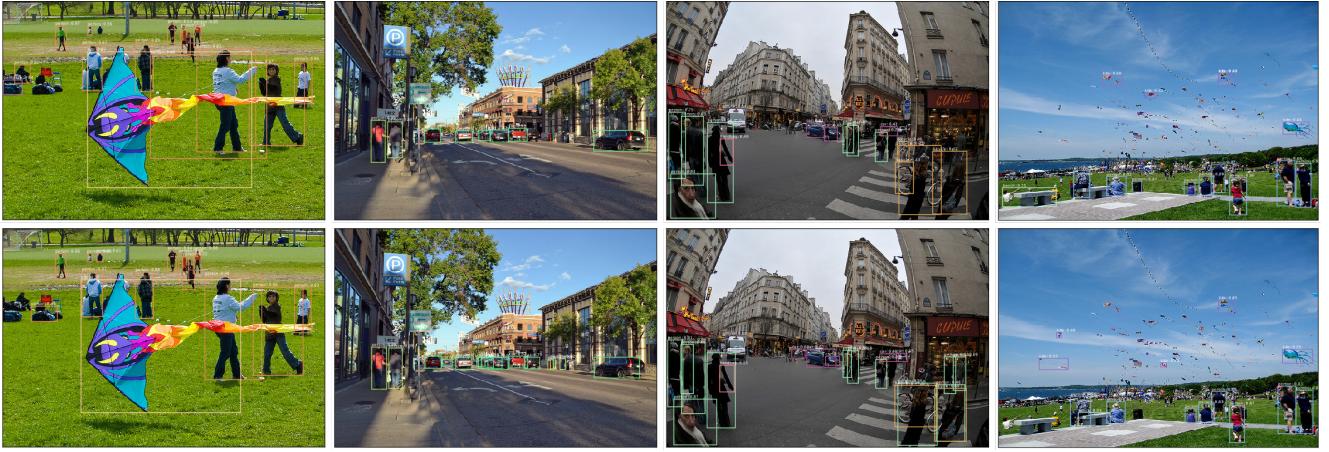


Figure 7. Some visualized detection results of TinyDet-S in the top row and TinyDet-M in the bottom row. Best view in PDF.

Table 4. Evaluation of backbone enhancement. The enhanced backbone significantly improves the detection performance, especially that of small objects.

| Backbone | MFLOPs | AP | AP ^s | AP ^m | AP ^l |
|-----------------------|--------|------|-----------------|-----------------|-----------------|
| MobileNetV2 [38] | 908 | 28.2 | 12.5 | 29.5 | 43.3 |
| SNet535 [34] | 1602 | 28.5 | 11.9 | 29.3 | 44.9 |
| FBNet-C [47] | 1055 | 27.8 | 11.6 | 28.6 | 43.9 |
| Proxyless-GPU [4] | 1304 | 27.8 | 11.8 | 28.4 | 44.3 |
| MobileNetV3-BC w/o SE | 988 | 29.5 | 14.2 | 31.1 | 44.1 |
| MobileNetV3 [14] | 963 | 28.4 | 13.0 | 29.6 | 43.7 |
| MobileNetV3-B | 1016 | 29.5 | 14.3 | 30.7 | 44.3 |
| MobileNetV3-C | 1001 | 29.8 | 14.6 | 30.9 | 44.8 |
| MobileNetV3-BC | 991 | 30.3 | 15.6 | 31.9 | 45.0 |

TinyDet (configurations in Tab. 1). Tab. 4 shows the comparison of different configurations. For the fair comparison, we adjust group numbers of TinyFPN and TinyRPN to keep the total computation budget similar (around 1 GFLOPs). Our MobileNetV3-BC outperforms MobileNetV3 and other variants. Increasing layer numbers and channel numbers respectively improves AP by 1.1% and 1.4%. When both are adopted, AP is improved by 1.9%. Besides, the gain on AP^s(+2.6) is more significant than AP^m(+1.3) and AP^l(+1.3). With more computation allocated to the early stages, detailed information is enhanced and benefits small object detection more.

We also compare MobileNetV3-BC with other representative lightweight backbones (Tab. 4). We remove SE module for fair comparison. MobileNetV3-BC achieves better results than both manually designed backbones (SNet535 [34] and MobileNetV2 [38]) and automatically searched backbones (FBNet [47] and Proxyless [4]).

Table 5. Results of applying different convolution operators in FPN and RPN. The proposed sparsely-connected convolution in FPN and RPN greatly reduces the computation cost and keeps comparable performance.

| Conv type | MFLOPs | AP | AP ⁵⁰ | AP ⁷⁵ |
|---------------------------|--------|------|------------------|------------------|
| Conventional Conv | 10115 | 31.8 | 52.5 | 33.5 |
| Depth-wise Separable Conv | 1970 | 30.8 | 51.5 | 32.4 |
| Sparsely-connected Conv | 991 | 30.3 | 50.9 | 32.1 |

4.3.2 Ablation Study on the SCConv in TinyFPN and TinyRPN

To make FPN and RPN lightweight, we combine the depth-wise convolution and point-wise group convolution as the sparsely-connected convolution (SCConv). As shown in Tab. 5, compared with the conventional convolution or depth-wise separable convolution [8, 15] in FPN and RPN, SCConv greatly reduces the computation cost and achieves comparable results. The results confirm that the SCConv in TinyFPN and TinyRPN causes little degradation on detection performance. For the lightweight detector design, adopting SCConv in detection heads is an effective method to save computation cost while maintaining high detection performance.

We also evaluate different group number settings of SCConvs in TinyFPN and TinyRPN. The results are presented in Tab. 6. From the comparison between row (c) and (e), we find that setting different group numbers for different pyramid levels is better than setting the same group number. A good practice is to use more groups for the bottom pyramids and fewer groups in the top pyramids in FPN (Tab. 6(e)).

Table 6. Evaluation of different group configurations. TinyFPN groups in the table respectively corresponds to pyramid levels with stride 4, 8, 16, 32 and 64. We adopt (e) in our TinyDet.

| TinyFPN groups | TinyRPN group | MFLOPs | AP | AP ⁵⁰ | AP ⁷⁵ |
|-----------------------------|---------------|--------|------|------------------|------------------|
| (a) 1, 1, 1, 1, 1 | 1 | 1970 | 30.8 | 51.5 | 32.4 |
| (b) 245, 245, 245, 245, 245 | 245 | 950 | 28.8 | 49.6 | 30.1 |
| (c) 7, 7, 7, 7, 7 | 49 | 1030 | 30.0 | 50.4 | 31.4 |
| (d) 1, 1, 5, 7, 49 | 49 | 1440 | 30.6 | 51.1 | 32.4 |
| (e) 49, 7, 5, 1, 1 | 49 | 991 | 30.3 | 50.9 | 32.1 |

4.3.3 Ablation Study on Feature Alignment

We evaluate how feature misalignment, described in Sec. 3.1.2, affects the detection performance in Tab. 7. By better aligning the backbone features using a simple average pooling, we get a 0.5% gain on AP. Note that the gain on AP^s (+1.4%) is higher than that on AP^m (+0.6%) or AP^l (+0.8%). This is because small objects are much more sensitive to feature misalignment and benefit more from this correction.

Table 7. Detection results without and with feature alignment. Alignment significantly improves detection performance of small objects.

| | AP | AP ⁵⁰ | AP ⁷⁵ | AP ^s | AP ^m | AP ^l |
|-------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|
| without alignment | 29.8 | 50.4 | 31.1 | 14.2 | 31.3 | 44.2 |
| with alignment | 30.3 _{±0.5} | 50.9 _{±0.5} | 32.1 _{±1.0} | 15.6 _{±1.4} | 31.9 _{±0.6} | 45.0 _{±0.8} |

Table 8. Inference latency of TinyDet on two types of ARM CPUs with single thread. No specific optimization is adopted.

| CPU type | Snapdragon 865 | Kirin 820 |
|-----------|----------------|-----------|
| TinyDet-S | 103ms | 133ms |
| TinyDet-M | 179ms | 236ms |
| TinyDet-L | 312ms | 386ms |

4.4. Limitation

There is a limitation in this study. Currently we cannot fairly compare the speed with other detectors. The reasons are two-fold. Firstly, the speed of a detector is highly dependent on the inference SDK (*e.g.*, Apple’s CoreML and Google’s TensorFlowLite). For ThunderNet, it uses a third-party high-performance inference SDK, which is not publicly available. This makes fair speed comparison in the same hardware condition infeasible. Secondly, most inference SDKs are mainly optimized for computation-intensive models and less focuses on the memory bandwidth. However, TinyDet is computation-efficient and requires bandwidth-oriented optimization. The inconsistency makes the TinyDet sub-optimal in speed. Currently we

offer the inference latency of TinyDet on two types of ARM CPUs with single thread and without any optimization (Tab. 8).

FLOPs is a widely-accepted metric of great generality, especially in the neural architecture search (NAS) domain. FLOPs can be calculated theoretically, irrelevant to specific hardware implementation. And according to the roofline theory [46], FLOPs determines the upper bound of speed. Considering the low computation requirement, with specific optimization, TinyDet can achieve excellent inference speed.

5. Conclusion

In this paper, we target at detecting generic (*i.e.*, 80 object categories) small (*i.e.*, $\leq (32 \times 32)/(480 \times 640) \approx 0.33\%$ area of the input image) objects in the computation constrained (*i.e.*, ≤ 1 GFLOPs) setting. The proposed technologies, such as high-resolution detection feature maps, TinyFPN/RPN, early-stage enhanced backbone and average pooling-based feature alignment are focusing on solving this problem. We obtain outstanding results - more than 100% AP improvement over the previous state-of-the-art lightweight generic detector on COCO small object detection. In the future, we would like to implement and optimize the proposed TinyDet on ASICs and edge devices.

References

- [1] Yancheng Bai, Yongqiang Zhang, Mingli Ding, and Bernard Ghanem. SOD-MTGAN: small object detection via multi-task generative adversarial network. In *ECCV*, 2018. [2](#)
- [2] Sean Bell, C. Lawrence Zitnick, Kavita Bala, and Ross B. Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *CVPR*, 2016. [2](#)
- [3] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. Soft-nms - improving object detection with one line of code. In *ICCV*, 2017. [7](#)
- [4] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. [8](#)
- [5] Guimei Cao, Xuemei Xie, Wenzhe Yang, Quan Liao, Guangming Shi, and Jinjian Wu. Feature-fused SSD: fast detection for small objects. *arXiv:1709.05054*, 2017. [2](#)
- [6] Chen Chen, Mengyuan Liu, Xiandong Meng, Wanpeng Xiao, and Qi Ju. Refinedetlite: A lightweight one-stage object detection framework for cpu-only devices. *arXiv:1911.08855*, 2019. [1, 2, 5](#)
- [7] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, and Junhai Ma. R-Net: A fast and accurate detector for small objects. In *CVPR*, 2020. [1, 2, 3, 4, 5, 6, 7](#)

According to the definition of small objects in the COCO dataset, referring to [25] and <http://cocodataset.org/#detection-eval>

- Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDection: Open mmlab detection toolbox and benchmark. *arXiv:1906.07155*, 2019. 6
- [8] François Fleuret. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017. 3, 5, 8
- [9] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: object detection via region-based fully convolutional networks. In *NIPS*, 2016. 2, 7
- [10] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C. Berg. DSSD : Deconvolutional single shot detector. *arXiv:1701.06659*, 2017. 2
- [11] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021. 7
- [12] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. NAS-FPN: learning scalable feature pyramid architecture for object detection. In *CVPR*, 2019. 7
- [13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014. 2
- [14] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *arXiv:1905.02244*, 2019. 2, 3, 4, 8, 11, 13, 14
- [15] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017. 2, 3, 5, 8
- [16] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018. 4, 13, 14
- [17] Peiyun Hu and Deva Ramanan. Finding tiny faces. In *CVPR*, 2017. 2
- [18] Glenn Jocher, Yonghye Kwon, guigarfr, perry0418, Josh Veitch-Michaelis, Ttayu, Daniel Suess, Fatih Baltaci, Gabriel Bianconi, IlyaOvodov, Marc, e96031413, Chang Lee, Dustin Kendall, Falak, Francisco Reveriano, FuLin, GoogleWiki, Jason Nataprawira, Jeremy Hu, LinCoce, LukeAI, NanoCode012, NirZarrabi, Oulbacha Reda, Piotr Skalski, SergioSanchezMontesUAM, Shiwei Song, Thomas Havlik, and Timothy M. Shead. ultralytics/yolov3: v9.5.0 - YOLOv5 v5.0 release compatibility update for YOLOv3, Apr. 2021. 7
- [19] Jianan Li, Xiaodan Liang, Yunchao Wei, Tingfa Xu, Jiashi Feng, and Shuicheng Yan. Perceptual generative adversarial networks for small object detection. In *CVPR*, 2017. 2
- [20] Yuxi Li, Jiuwei Li, Weiyao Lin, and Jianguo Li. Tiny-dsod: Lightweight object detection for resource-restricted usages. In *BMVC*, 2018. 7
- [21] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Light-head R-CNN: in defense of two-stage object detector. *arXiv:1711.07264*, 2017. 2, 7
- [22] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: Design backbone for object detection. In *The European Conference on Computer Vision (ECCV)*, September 2018. 2
- [23] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 1, 2, 7
- [24] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017. 2
- [25] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *ECCV*, 2014. 2, 6, 9
- [26] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall F. Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *CVPR*, 2015. 3
- [27] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. In *ECCV*, 2016. 2, 3, 7
- [28] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv:1608.03983*, 2016. 6
- [29] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard S. Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *NIPS*, 2016. 4
- [30] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet V2: practical guidelines for efficient CNN architecture design. In *ECCV*, 2018. 2, 3
- [31] Junhyug Noh, Wonho Bae, Wonhee Lee, Jinhwan Seo, and Gunhee Kim. Better to follow, follow to be better: Towards precise supervision of feature super-resolution for small object detection. In *ICCV*, 2019. 2
- [32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NIPS*, 2019. 6
- [33] Chao Peng, Tete Xiao, Zeming Li, Yuning Jiang, Xiangyu Zhang, Kai Jia, Gang Yu, and Jian Sun. Megdet: A large mini-batch object detector. In *CVPR*, 2018. 7
- [34] Zheng Qin, Zeming Li, Zhaoning Zhang, Yiping Bao, Gang Yu, Yuxing Peng, and Jian Sun. Thundernet: Towards real-time generic object detection on mobile devices. In *ICCV*, 2019. 1, 2, 5, 7, 8
- [35] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *CVPR*, 2017. 7
- [36] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv:1804.02767*, 2018. 2, 7
- [37] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *NIPS*, 2015. 1, 2, 5, 6, 7
- [38] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 1, 2, 3, 4, 5, 7, 8, 13, 14
- [39] Abhinav Shrivastava, Abhinav Gupta, and Ross B. Girshick. Training region-based object detectors with online hard example mining. In *CVPR*, 2016. 7

- [40] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 2
- [41] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. *arXiv:1911.09070*, 2019. 1, 2, 5, 7
- [42] Chien-Yao Wang, Hong-Yuan Mark Liao, Ping-Yang Chen, and Jun-Wei Hsieh. Enriching variety of layer-wise learning information by gradient combination. In *ICCVW*, 2019. 7
- [43] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. CspNet: A new backbone that can enhance learning capability of CNN. In *CVPRW*, 2020. 7
- [44] Jun Wang, Tanner A. Bohn, and Charles X. Ling. Pelee: A real-time object detection system on mobile devices. In *NIPS*, 2018. 1, 2, 5, 7
- [45] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. Deep high-resolution representation learning for visual recognition. *arXiv:1908.07919*, 2019. 2
- [46] Samuel Williams, Andrew Waterman, and David A. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 2009. 9
- [47] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019. 8
- [48] Wei Xiang, Dong-Qing Zhang, Heather Yu, and Vassilis Athitsos. Context-aware single-shot detector. In *WACV*, 2018. 2
- [49] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018. 2, 3, 5

A. Appendix

A.1. Overall Computation Allocation of TinyDets

Tab. 9 provides the overall computation allocation of TinyDets. The backbone network contains most of the computation budget ($\sim 70\%$) while TinyFPN, TinyRPN and R-CNN are extremely computationally cheap.

Table 9. Computation allocation of different parts (MFLOPs).

| Detector | Total | Backbone | TinyFPN | TinyRPN | R-CNN |
|-----------|-------------|------------|-----------|----------|----------|
| TinyDet-S | 495 (100%) | 347 (70%) | 64 (13%) | 16 (3%) | 68 (14%) |
| TinyDet-M | 991 (100%) | 703 (71%) | 155 (16%) | 65 (7%) | 68 (7%) |
| TinyDet-L | 2427 (100%) | 1797 (74%) | 396 (16%) | 166 (7%) | 68 (3%) |

A.2. Detailed Backbone Information

TinyDet-S is within the computation constraint of 500 MFLOPs. It takes a more lightweight variant of MobileNetV3 [14] as the backbone, termed as MobileNetV3-D. The specification of MobileNetV3-D can be found in Tab. 10. To reduce computation cost, we remove the detailed information enhancement used in TinyDet-M and TinyDet-L, and shrink the expansion size of high-level layers.

TinyDet-M is within the computation constraint of 1000 MFLOPs. It takes a MobileNetV3-BC as the backbone (Tab. 11). The input resolution is 320×320 .

TinyDet-L is targeted at a high accuracy. It also adopts MobileNetV3-BC as the backbone but takes a larger input resolution as 512×512 (Tab. 12).

A.3. Computation Cost of TinyFPN & TinyRPN

In TinyDet-S, different from the structure shown in Fig. 4 of the main paper, we drop the highest-resolution pyramid level (stride=4) to reduce computation cost. TinyFPN contains four 1×1 convs and four SCConvs (Tab. 13). TinyRPN contains four SCConvs, four convs for predicting scores and four convs for box regression ((Tab. 14).

As shown in Fig. 4 of the main paper, TinyFPN contains five 1×1 convs and five SCConvs (Tab. 15). TinyRPN contains five SCConvs, five convs for predicting scores and five convs for box regression (Tab. 16).

TinyFPN and TinyRPN of TinyDet-L have the same structure with those of TinyDet-M, except the resolution. TinyFPN contains five 1×1 convs and five SCConvs (Tab. 17). TinyRPN contains five SCConvs, five convs for predicting scores and five convs for regression (Tab. 18).

A.4. Computation Cost of R-CNN

The R-CNN head is based on PSRoI align, which consists of three fully connected layers, as shown in Fig. 8. 200 RoIs are used in inference phase. Based on 200 RoIs, the

computation cost of R-CNN is:

$$\begin{aligned} & 200 \times (245 \times 1024 + 1024 \times 81 + 1024 \times 4) \\ & = 67.584 \times 10^6 \\ & \approx 68 \text{ MFLOPs} \end{aligned} \quad (1)$$

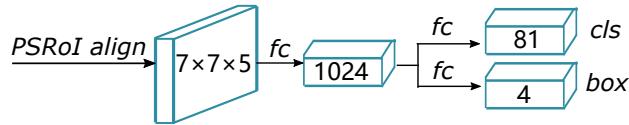


Figure 8. Structure of R-CNN head.

Table 12. Specification for the backbone of TinyDet-L. Bneck denotes the inverted residual bottleneck structure [38]. ExSize denotes expansion size. SE denotes the squeeze-and-excitation module [16]. NL denotes the type of the used nonlinearity. HS denotes h-swish [14] and RE denotes ReLU. FPN denotes whether the output of the block is fed into FPN. Enhancement denotes modifications compared with the original MobileNetV3 and "c" denotes channel.

| Input | Operator | ExSize | Out | SE | NL | Stride | FPN | Enhancement | MFLOPs |
|-------------------|----------------------|--------|-----|----|----|--------|-----|--------------|---------|
| $512^2 \times 3$ | conv2d, 3×3 | - | 24 | - | HS | 2 | - | c + 50% | 42.47 |
| $256^2 \times 24$ | bneck, 3×3 | 24 | 24 | - | RE | 1 | - | c + 50% | 51.91 |
| $256^2 \times 24$ | bneck, 3×3 | 72 | 36 | - | RE | 2 | - | c + 50% | 166.33 |
| $128^2 \times 36$ | bneck, 3×3 | 108 | 36 | - | RE | 1 | - | added blocks | 143.33 |
| $128^2 \times 36$ | bneck, 3×3 | 108 | 36 | - | RE | 1 | ✓ | added blocks | 143.33 |
| $128^2 \times 36$ | bneck, 3×3 | 108 | 36 | - | RE | 1 | ✓ | c + 50% | 143.33 |
| $128^2 \times 36$ | bneck, 5×5 | 108 | 60 | ✓ | RE | 2 | - | c + 50% | 101.31 |
| $64^2 \times 60$ | bneck, 5×5 | 180 | 60 | ✓ | RE | 1 | - | c + 50% | 106.92 |
| $64^2 \times 60$ | bneck, 5×5 | 180 | 60 | ✓ | RE | 1 | ✓ | c + 50% | 106.92 |
| $64^2 \times 60$ | bneck, 3×3 | 240 | 80 | - | HS | 2 | - | - | 80.86 |
| $32^2 \times 80$ | bneck, 3×3 | 200 | 80 | - | HS | 1 | - | - | 34.61 |
| $32^2 \times 80$ | bneck, 3×3 | 184 | 80 | - | HS | 1 | - | - | 31.84 |
| $32^2 \times 80$ | bneck, 3×3 | 184 | 80 | - | HS | 1 | - | - | 31.84 |
| $32^2 \times 80$ | bneck, 3×3 | 480 | 112 | ✓ | HS | 1 | - | - | 98.91 |
| $32^2 \times 112$ | bneck, 3×3 | 672 | 112 | ✓ | HS | 1 | ✓ | - | 160.56 |
| $32^2 \times 112$ | bneck, 5×5 | 672 | 160 | ✓ | HS | 2 | - | - | 109.12 |
| $16^2 \times 160$ | bneck, 5×5 | 960 | 160 | ✓ | HS | 1 | - | - | 85.25 |
| $16^2 \times 160$ | bneck, 5×5 | 960 | 160 | ✓ | HS | 1 | ✓ | - | 85.25 |
| $16^2 \times 160$ | bneck, 5×5 | 960 | 160 | ✓ | HS | 2 | - | - | 51.15 |
| $8^2 \times 160$ | bneck, 5×5 | 960 | 160 | ✓ | HS | 1 | ✓ | - | 21.66 |
| In total | - | - | - | - | - | - | - | - | 1796.90 |

Table 13. Computation cost of TinyFPN of TinyDet-S.

| Operator | Input | Output | MFLOPs |
|--------------------|-------------------|-------------------|--------|
| Conv, 1×1 | $40^2 \times 40$ | $40^2 \times 245$ | 16.07 |
| Conv, 1×1 | $20^2 \times 112$ | $20^2 \times 245$ | 11.01 |
| Conv, 1×1 | $10^2 \times 160$ | $10^2 \times 245$ | 3.95 |
| Conv, 1×1 | $5^2 \times 160$ | $5^2 \times 245$ | 0.99 |
| SCConv, g=7 | $40^2 \times 245$ | $40^2 \times 245$ | 18.03 |
| SCConv, g=5 | $20^2 \times 245$ | $20^2 \times 245$ | 5.88 |
| SCConv, g=1 | $10^2 \times 245$ | $10^2 \times 245$ | 6.27 |
| SCConv, g=1 | $5^2 \times 245$ | $5^2 \times 245$ | 1.57 |
| In total | - | - | 63.77 |

Table 14. Computation cost of TinyRPN of TinyDet-S.

| Operator | Input | Output | MFLOPs |
|----------------------------|-------------------|-------------------|--------|
| SCConv, g=49 | $40^2 \times 245$ | $40^2 \times 245$ | |
| | $20^2 \times 245$ | $20^2 \times 245$ | |
| | $10^2 \times 245$ | $10^2 \times 245$ | 8.33 |
| | $5^2 \times 245$ | $5^2 \times 245$ | |
| Conv, 1×1 (score) | $40^2 \times 245$ | $40^2 \times 3$ | |
| | $20^2 \times 245$ | $20^2 \times 3$ | |
| | $10^2 \times 245$ | $10^2 \times 3$ | 1.57 |
| | $5^2 \times 245$ | $5^2 \times 3$ | |
| Conv, 1×1 (box) | $40^2 \times 245$ | $40^2 \times 12$ | |
| | $20^2 \times 245$ | $20^2 \times 12$ | |
| | $10^2 \times 245$ | $10^2 \times 12$ | 6.27 |
| | $5^2 \times 245$ | $5^2 \times 12$ | |
| In total | - | - | 16.17 |

Table 15. Computation cost of TinyFPN of TinyDet-M.

| Operator | Input | Output | MFLOPs |
|--------------------|-------------------|-------------------|--------|
| Conv, 1×1 | $80^2 \times 36$ | $80^2 \times 245$ | 58.02 |
| Conv, 1×1 | $40^2 \times 60$ | $40^2 \times 245$ | 23.91 |
| Conv, 1×1 | $20^2 \times 112$ | $20^2 \times 245$ | 11.07 |
| Conv, 1×1 | $10^2 \times 160$ | $10^2 \times 245$ | 3.95 |
| Conv, 1×1 | $5^2 \times 160$ | $5^2 \times 245$ | 0.99 |
| SCConv, g=49 | $80^2 \times 245$ | $80^2 \times 245$ | 25.09 |
| SCConv, g=7 | $40^2 \times 245$ | $40^2 \times 245$ | 18.03 |
| SCConv, g=5 | $20^2 \times 245$ | $20^2 \times 245$ | 5.88 |
| SCConv, g=1 | $10^2 \times 245$ | $10^2 \times 245$ | 6.27 |
| SCConv, g=1 | $5^2 \times 245$ | $5^2 \times 245$ | 1.57 |
| In total | - | - | 154.78 |

Table 16. Computation cost of TinyRPN of TinyDet-M.

| Operator | Input | Output | MFLOPs |
|----------------------------|-------------------|-------------------|--------|
| SCConv, g=49 | $80^2 \times 245$ | $80^2 \times 245$ | |
| | $40^2 \times 245$ | $40^2 \times 245$ | |
| | $20^2 \times 245$ | $20^2 \times 245$ | 33.42 |
| | $10^2 \times 245$ | $10^2 \times 245$ | |
| | $5^2 \times 245$ | $5^2 \times 245$ | |
| Conv, 1×1 (score) | $80^2 \times 245$ | $80^2 \times 3$ | |
| | $40^2 \times 245$ | $40^2 \times 3$ | |
| | $20^2 \times 245$ | $20^2 \times 3$ | 6.29 |
| | $10^2 \times 245$ | $10^2 \times 3$ | |
| | $5^2 \times 245$ | $5^2 \times 3$ | |
| Conv, 1×1 (box) | $80^2 \times 245$ | $80^2 \times 12$ | |
| | $40^2 \times 245$ | $40^2 \times 12$ | |
| | $20^2 \times 245$ | $20^2 \times 12$ | 25.17 |
| | $10^2 \times 245$ | $10^2 \times 12$ | |
| | $5^2 \times 245$ | $5^2 \times 12$ | |
| In total | - | - | 64.88 |

Table 17. Computation cost of TinyFPN of TinyDet-L.

| Operator | Input | Output | MFLOPs |
|--------------------|--------------------|--------------------|--------|
| Conv, 1×1 | $128^2 \times 36$ | $128^2 \times 245$ | 148.52 |
| Conv, 1×1 | $64^2 \times 60$ | $64^2 \times 245$ | 61.22 |
| Conv, 1×1 | $32^2 \times 112$ | $32^2 \times 245$ | 28.35 |
| Conv, 1×1 | $16^2 \times 160$ | $16^2 \times 245$ | 10.10 |
| Conv, 1×1 | $8^2 \times 160$ | $8^2 \times 245$ | 2.52 |
| SCConv, g=49 | $128^2 \times 245$ | $128^2 \times 245$ | 64.23 |
| SCConv, g=7 | $64^2 \times 245$ | $64^2 \times 245$ | 46.16 |
| SCConv, g=5 | $32^2 \times 245$ | $32^2 \times 245$ | 15.05 |
| SCConv, g=1 | $16^2 \times 245$ | $16^2 \times 245$ | 16.06 |
| SCConv, g=1 | $8^2 \times 245$ | $8^2 \times 245$ | 4.01 |
| In total | - | - | 396.22 |

Table 18. Computation cost of TinyRPN of TinyDet-L.

| Operator | Input | Output | MFLOPs |
|----------------------------|--------------------|--------------------|--------|
| SCConv, g=49 | $128^2 \times 245$ | $128^2 \times 245$ | |
| | $64^2 \times 245$ | $64^2 \times 245$ | |
| | $32^2 \times 245$ | $32^2 \times 245$ | 85.55 |
| | $16^2 \times 245$ | $16^2 \times 245$ | |
| | $8^2 \times 245$ | $8^2 \times 245$ | |
| Conv, 1×1 (score) | $128^2 \times 245$ | $128^2 \times 3$ | |
| | $64^2 \times 245$ | $64^2 \times 3$ | |
| | $32^2 \times 245$ | $32^2 \times 3$ | 16.11 |
| | $16^2 \times 245$ | $16^2 \times 3$ | |
| | $8^2 \times 245$ | $8^2 \times 3$ | |
| Conv, 1×1 (box) | $128^2 \times 245$ | $128^2 \times 12$ | |
| | $64^2 \times 245$ | $64^2 \times 12$ | |
| | $32^2 \times 245$ | $32^2 \times 12$ | 64.42 |
| | $16^2 \times 245$ | $16^2 \times 12$ | |
| | $8^2 \times 245$ | $8^2 \times 12$ | |
| In total | - | - | 166.08 |